

18p

AUTOMATIC ESTIMATES OF COMPUTATIONAL ERRORS

By G. J. Moshos and L. R. Turner

ABSTRACT

NASA TMX 51660

~~NASA CR 52672~~

A new technique has been designed to aid the programmer in accounting **A** for computational errors, that is, rounding off and truncation errors in computational programs. This scheme is one in which signed errors based on higher accuracy calculations rather than error bounds are calculated. The discussion includes a formulation of techniques for obtaining realistic error estimates for a wide range of computational problems and methods for reporting these errors to the programmer in an analysis of error. A series of test programs has been run in this arithmetic mode to obtain initial experience. This experience is reported, as well as a curious effect when the problem has more than one solution.

I.E.E.E. Session on Error-
Trace Arithmetic
Chicago, Illinois
Oct. 28-30, 1963

18p

FACILITY FORM 602	N 65 88490	
	(ACCESSION NUMBER)	(THRU)
	18	None
	(PAGES)	(CODE)
	TMX 51660	
	(NASA CR OR TMX OR AD NUMBER)	(CATEGORY)

~~TMX 51660~~

AUTOMATIC ESTIMATES OF COMPUTATIONAL ERRORS

By G. J. Moshos* and L. R. Turner*

SUMMARY


E-2199

A new technique has been designed to account for computational errors, that is, rounding off and truncation errors in computational programs. Each number of the system is represented by two floating-point numbers, one representing the nominal precision result, and the other representing the signed accumulated error based on higher accuracy calculations. The discussion includes a formulation of techniques for obtaining realistic error estimates for a wide range of computational problems and methods for reporting these errors to the programmer to aid in an analysis of error. Finally, the experience achieved by using this arithmetic in several test problems, which were selected to cover a variety of difficulties, is reported.

INTRODUCTION

The support of research programs of a scientific center often involves fresh computational problems for which the nature of the solution, or a guaranteed method of achieving it, is not known. Assuming that such a problem can be formulated and a possible computational procedure devised, if the answers are to be meaningfully interpreted, the computational program must then be assessed with an analysis of the various errors introduced by the program and data. This type of analysis is especially difficult to perform, since it usually involves a greater complexity of analysis

*NASA Lewis Research Center, Cleveland, Ohio.



than the program itself. Moreover, this analysis is further complicated by the use of floating-point arithmetic. As a consequence of the complexity of error analysis of problems in floating-point arithmetic, some automatic technique is needed to aid the analysis of such problems. The automatic techniques that have been proposed in the literature are based on either performing the calculations exclusively with significant digits or calculating an error bound along with the nominal precision result. Since the estimate of error in each of these arithmetics is arrived at by accuracy reducing calculations, the estimate of error can be expected to be pessimistic.

The purpose of this paper is to discuss a new technique that is designed to account for computational errors, that is, rounding-off and truncation errors. This technique is one in which a signed error rather than an error bound is calculated. In order to achieve realistic error estimates, the error estimates are based on higher accuracy calculations. Methods are presented for obtaining estimates of errors in arithmetic operations, in nonalgebraic function evaluation, and in iterative processes.

Because there are two answers at each point in the computations, branching on value may become indecisive. This indecisiveness is a troublesome feature in the management of a problem, but it may also be an invaluable aid in debugging. This use of error arithmetic and others are discussed.

Finally, the experience that was gained by running several problems in this arithmetic mode, including a curious effect when the problem had more than one solution, is discussed.

ARITHMETIC WITH SIGNED ERROR

A scheme that is proposed at the Lewis Research Center is one in which signed errors, rather than error bounds, are calculated. Each number N of the system is represented by two floating-point numbers (n, E_n) . During the arithmetic operations, n represents a nominal precision result, while E_n represents the accumulated error which consists of the generated error of the operation and error propagated from previous calculations. With (a, E_a) and (b, E_b) assumed as the two operands and (c, E_c) the result of an arithmetic operation, the computed error estimate E_c is then calculated by the formulas in Table I to the precision permitted by the machine representation of E_c . (The bracketed quantities in +, -, and \times are the generated errors.)

TABLE I. - FORMULAS FOR COMPUTING ESTIMATE OF TOTAL ERROR

Operation, op	Computed error estimate, E_c
+	$[(a + b) - c] + E_a + E_b$
-	$[(a - b) - c] + E_a - E_b$
\times	$[ab - c] + aE_b + bE_a + E_aE_b$
\div	$\frac{(a + E_a) - c(b + E_b)}{(b + E_b)}$

Each step of this arithmetic is equivalent to that of performing the computations simultaneously with a nominal precision arithmetic and an extended precision arithmetic and computing the estimate of error by taking the difference of the two values. The formulas in Table I represent accurate estimates of the error in the nominal precision result

$c = a \text{ op } b$. Attention is called to the fact that in multiplication $E_a E_b$ is insignificant if the errors are small, but will not be negligible if the errors are large. In division, $a + E_a - c(b + E_b)$, where c is the nominal precision result, is an accurate remainder, and $b + E_b$ can be used effectively as the sum of b and its error reduced to the representation of the nominal precision result.

A typical result of a simple arithmetic operation is shown by the decimal addition $1/7 + 8/9$, where $1/7 \doteq (0.142857, 0.142857 \times 10^{-6})$ and $8/9 \doteq (0.888889, -0.111111 \times 10^{-6})$. Their sum, $65/63$, has the representation $65/63 \doteq (1.03175, -0.396825 \times 10^{-5})$. The error -0.396825×10^{-5} is made up of two parts - the generated error $[(a + b) - c] = 0.888889 + 0.142857 - 1.03175 = -0.4 \times 10^{-5}$ and the propagated error $E_a + E_b = 0.142857 \times 10^{-6} - 0.111111 \times 10^{-6} = 0.31746 \times 10^{-7}$. The total computed error rounded to six significant decimal digits is -0.396825×10^{-5} .

A second example, the product of $13/9$ and $5/7$ has as its input the number pairs $(1.44444, +0.444444 \times 10^{-5})$ and $(0.714286, -0.285714 \times 10^{-6})$, which yield the pair $(1.03174, 0.603174 \times 10^{-5})$, whether the product $E_a E_b$ is included or not, and whether the low order part of the product is rounded or not. The product $E_a E_b$ is generally not important if the errors are small, but because errors can and do become large, an accurate error trace requires the consideration of this term through some artifice.

The subtraction $11/12 - 10/11$ (without detail) leads to the pair $(0.757600 \times 10^{-2}, -0.242424 \times 10^{-6})$, which illustrates that digit positions of an error estimate may overlap the digit positions of the nominal precision result. It is similarly possible for the ranges of digit positions

to be disjoint. When the errors are small, $c + E_c$ is very nearly equal to the result that would have been obtained by the systematic use of extended precision.

ESTIMATE OF ERROR IN THE EVALUATION OF
NONALGEBRAIC FUNCTIONS

Throughout this discussion we have in mind that two results are available in the computer after each calculation. One result is arrived at through nominal precision arithmetic, and the other is a more accurate standard for gaging the nominal precision result. If (a, E_a) is the argument in the evaluation of the nonalgebraic function $f(x)$, a represents the nominal precision number and $(a + E_a)$ represents a high precision calculated standard. Given some procedure $\bar{f}(x)$ for approximating the function $f(x)$ in nominal precision arithmetic, the standard for comparison must be arrived at through higher accuracy calculations, which must account for the generated error in the approximate procedure $\bar{f}(x)$, as well as the error propagated from previous calculations. Consequently, the error E_f may be equivalently defined as the sum of the propagated and generated errors, which may be expressed as $f(a + E_a) - f(a)$ and $f(a) - \bar{f}(a)$, respectively, which are the same as $E_f = f(a + E_a) - \bar{f}(a)$.

The value of the function employed in nominal precision arithmetic is customarily obtained by some approximation that represents a good compromise between accuracy and computer speed. For example, such an approximation might be a Tchebycheff polynomial approximation. If E_a is large, this procedure could be used to evaluate $f(a + E_a)$ to obtain the error E_f . However, if E_a is small, this approximation would fall short of

the goal, since truncation error would represent an important contribution to the total error and would not be reflected by the algorithm. A safe procedure for calculating $f(a + E_a)$ is to have available a high-accuracy routine based on an approximation such as a Taylor series or continued fractions that can achieve any necessary degree of accuracy.

ESTIMATE OF ERROR IN ITERATIVE PROCESSES

The error in an iterative process is defined as the difference between the asymptotic value of the answer and the machine value after the k^{th} iteration, that is, $x_\infty - \bar{x}_k$. Ordinarily, the error of the iterative process (as well as the rate of convergence) during computations is measured by a value computed by taking the difference of the machine values of two successive iterations, that is, $\bar{x}_{k+1} - \bar{x}_k$. When the error of the iterative process $x_\infty - \bar{x}_k$ is expanded by $(x_\infty - x_{k+1}) + (x_{k+1} - \bar{x}_{k+1}) + (\bar{x}_{k+1} - \bar{x}_k)$, it is noted that the measured error of an iterative process is further vitiated by the iterative truncation error $(x_\infty - x_{k+1})$. Consequently, in order to estimate realistically both the iterative truncation error and the rounding error, it is necessary to realize an accurate value of the asymptotic value of the answer with higher precision arithmetic. This may be achieved by evaluating the iterative process with error arithmetic, as discussed in the previous two sections.

The more efficient use of the computer may be realized in some large iterative processes by understanding when the error trace needs to be initiated. For the class of iterative processes of finite iterations, such as the method of conjugate gradient or Hestene's method of biorthogonalization, an estimate of error can be obtained only if the error trace

is started at the start of the iteration. In this respect, then, these iterative processes must be treated as direct or explicit evaluations.

For iterative processes that yield the answer only in the limit, the error trace need not be started at the start of the process. Stationary processes, such as the methods of successive overrelaxation, are perhaps the easiest to error trace, since the error trace may be started at any time in the process. Nevertheless, after the error trace is started, the process must be permitted to continue until both the nominal precision result and the estimate of error have converged. Asymptotically stationary processes, such as the Tchebycheff semi-iterative methods, which are asymptotically similar to the successive overrelaxation methods, may be treated like stationary processes.

A special problem arises when the measured error of the process is used to establish a strategy as, for example, in the use of Kulrud's algorithm for determining the optimal overrelaxation parameter. In this process, if convergence is delayed because of rounding-off errors, the Kulrud's algorithm indicates a larger than optimal relaxation parameter to be used with the error trace. The correct procedure in this case would be either to deactivate the parameter calculation or to base it only on the higher precision results.

BRANCHING

Branching in error arithmetic has an added complexity because the branchings, such as branching on value, can be based on either of two precisions. A potential source of difficulty in a program occurs when a branch is undecidable; that is, if x calls for one choice and $x + E_x$ calls for the other choice. The effect of a choice of a particular

direction of branching is difficult to predict. If the two options execute essentially the same data storage and control operations and, therefore, lead to the same terminal program step, the dichotomy is probably unimportant and is, therefore, called immaterial; otherwise it is called a material dichotomy.

The execution of a program that contains only immaterial dichotomies should lead to a good estimate of errors, if adequate account is taken of iterative and functional truncation errors.

The only assured method of obtaining good error estimates when branching is doubtful is to execute the problem twice, the first time basing all branching on the nominal precision results, and the second time basing the branching on the sum of the nominal precision results and the estimate of error.

A report of the occurrence of dichotomies during these two executions of the problem could be of considerable aid in the debugging of a doubtful problem.

USES OF ERROR ARITHMETIC

The current practice in most computer installations is to compile a program to run in single precision arithmetic. After the program has been run in this mode and the solution is found to be in doubt, the program is run in a double precision arithmetic. The programmer, however, is still left without an estimate of error. The primary motivation for the arithmetic proposed in this paper is to give the programmer an estimate of error to aid in an error analysis. Moreover, since the error is calculated on a higher precision basis, the number $x + E_x$ represents a higher

accuracy number. The error arithmetic then gives the programmer a much more informative result than that obtainable from double precision calculations.

The first task of a programmer trying to improve a solution that is unsatisfactory is to find where in the program a detailed analysis would be profitable. Since the error arithmetic can cope with both large and small errors, it can be used to trace the errors on a dynamic basis and locate code sequences for further study.

Improved performance in the solution of large problems can usually be accomplished by (1) local improvement, (2) extended precision calculations, and (3) complete reprogramming.

Local improvements can be made only up to the minimum unavoidable error that is permitted by the precision of the arithmetic being used. Variations in algorithms may be studied with error arithmetic with an aim at achieving minimum error. Moreover, it is entirely feasible that critical steps in the arithmetic be evaluated by several algorithms using error arithmetic, and that the result for further calculation be selected on a dynamic basis to achieve the smallest error.

If a higher precision is needed than is permitted by the least count of the machine representation, error arithmetic can again be utilized to determine to what extent this would be helpful.

When both of these techniques fail, reprogramming may provide an answer. In this respect, a combination of algorithms may be a feasible approach; that is, one algorithm to achieve a maximum rate of convergence and another to achieve a minimum error.

A potential source of difficulty in error tracing a computational program occurs when a choice in branching is undecidable. One option is to follow the branching determined by the nominal precision arithmetic with an aim at reporting the accrued error in each quantity that was originally specified for output and reporting instances in which the choice in branching is altered when the higher accuracy calculations are considered.

If branching dichotomy develops when the program is traced with this option, it should be traced again following the branchings determined by the higher precision results. If the paths determined by these two options do not terminate to the same program statement, it is reasonable to assume that the program is in trouble, and that the calculations are sensitive to precision. For example, the higher precision results may lead to an alarm, while the nominal precision results lead to an answer that appears reasonable. If the paths determined by these two options do terminate to the same program statement, the answers produced by the two options may be compared to aid the programmer in ascertaining if the branch was material or immaterial. If the two results are appreciably different, the branch may again be classified as material, and the program can again be ascertained to be in trouble. Unfortunately, if the two results are about the same, the branch cannot be definitely classified (although the probability is that it is immaterial). By programming a sufficient number of intermediate reports, however, the programmer can localize the trouble and through further analysis classify the branch.

A point should be made at this time regarding the error arithmetics that have been proposed in the literature. These arithmetics, except

possibly for range arithmetic, do not permit a simple procedure to trace the alternate paths when a branch becomes undecidable. In the case of range arithmetic, while the procedure can be formulated, the report in any precision sensitive problem can be expected to be pessimistic, and this debugging aid can no longer be applied.

STYLES OF REPORTING ERRORS

If the error arithmetic is to aid the programmer in the analysis of error, the output report must be such as to facilitate this analysis. The generated error, in general, will tend to be small; however, because small errors can be magnified by propagation, the styles of reporting should also permit the study of large errors. Consequently, the programmer would want to study the errors separately from the higher precision results in order to trace the propagation of errors in the machine representation of the results.

Several styles could then be used for the report. Since each style must be easily identified, however, the report must also contain the identification of the style. Experience with Lewis Research Center error arithmetic has indicated that four different styles would be useful.

Since the errors in any local algorithm would, in general, be small, the natural unit for measuring error is with respect to the minimum unavoidable error. In this way, a programmer may relate the algorithm to the best obtainable with the working precision of the machine. The first style of error reporting is, therefore, in terms of the least count of the principal part of the number.

When the numbers being reported represent a vector, some common basis for relating the individual errors to the vector is needed. Some vector

norm could properly be used for this comparison. As an example, the vector

$$\begin{pmatrix} Y_n, E_{Y_n} \\ Y'_n, E_{Y'_n} \end{pmatrix}$$

represents the solution and the first derivative of the differential equation $Y'' + Y = 0$, $Y(0) = 1$, and $Y'(0) = 0$. In this example, when the value of Y_n is near 1, the value of Y'_n is near zero. Reporting the errors with respect to the least count of the principal part of the number could be misleading. In any case, a much more informative comparison could be made by reporting the errors in terms of the least count of the machine representation of 1, which is the Euclidean norm for this vector. If the vector

$$\begin{pmatrix} Y_0, E_{Y_0} \\ Y_1, E_{Y_1} \\ \vdots \\ Y_n, E_{Y_n} \end{pmatrix}$$

is to be reported for this differential equation, then 1.0 again would suffice, since it represents the Gershgorin norm for this vector. Another style of reporting the error is in terms of the least count relative to some calculated or prespecified number.

If the internal representation and the report representation of the number were on the same number base, the previous two reports would suffice for reporting the errors. However, if the two representations were different, for example, binary and decimal, respectively, some side calculation

would be necessary to compare the error with the number on an absolute scale. Consequently, another useful style of reporting is a report of the signed error on an absolute scale.

Finally, the error associated with a number could be of no use to the error analysis, but what would be desired is a report of only the nominal precision number with no error report. Not only would this save machine time in not publishing an error report when none is necessary, but it would also aid the programmer in minimizing the reports that are to be studied.

EXPERIENCE WITH ERROR ARITHMETIC

The purpose of the first series of experiments run with this arithmetic was to obtain some initial experience with the ability of the arithmetic to predict error estimates. Since the objective was to test the arithmetic, the examples were chosen so as to exclude truncation errors from the demonstration. In order to accomplish this, the standard for comparison was based only on the rational operations that were employed. This was obtained either by a formal solution, or a program that was run with a higher precision arithmetic. Specific iterative procedures were chosen so that the starting value could be modified and so that it could be observed if the error estimate was properly adjusted. The programs, problems, and modifications used to obtain this initial experience were the following:

Program: Matrix inversion by biorthogonalization

Problems: (1) Well conditioned 4 by 4 matrix consisting of integer
elements

(2) Hilbert matrices of order 3, 4, and 5

Modifications: The results of the first iteration, that is, (r, E_r) were adjusted as follows

(1) Unmodified

(2) $x := r, E_x := 0$

(3) $x := r + E_r, E_x := 0$

and then used as starting values (x, E_x) for the second iteration.

Program: Integration of second order differential equations by an implicit three-point procedure

Problems: (1) $\frac{d^2y}{dx^2} + y = 0, y(0) = 1, \text{ and } y'(0) = 0$

$0 \leq x \leq 7$ for step sizes $h = 1, 1/2, 1/4, 1/8,$
 $1/16, 1/32, 1/64$

(2) Bessel's equation of index zero

$$\frac{d^2y}{dx^2} + \frac{1}{x} \frac{dy}{dx} + y = 0$$

$y(0) = 1, y'(0) = 0, \text{ and } y''(0) = -1/2$

$0 \leq x \leq 7$ for step sizes $h = 1, 1/2, 1/4, 1/8,$
 $1/16, 1/32, 1/64$

Modifications: The results of the first iteration, that is, (r, E_r) , were adjusted as follows

(1) Unmodified

(2) $x := r, E_x := 0$

(3) $x := r + E_r, E_x := 0$

and then used as starting values (x, E_x) for the second iteration.

In all these cases the error arithmetic calculated an answer and gave a prediction of error. Moreover, in most cases, the prediction of error

was excellent. It is interesting to note that in these examples an estimate of error was obtained independently of whether or not an error was assumed in the starting value.

The next series of experiments was run to demonstrate some special problems with error arithmetic.

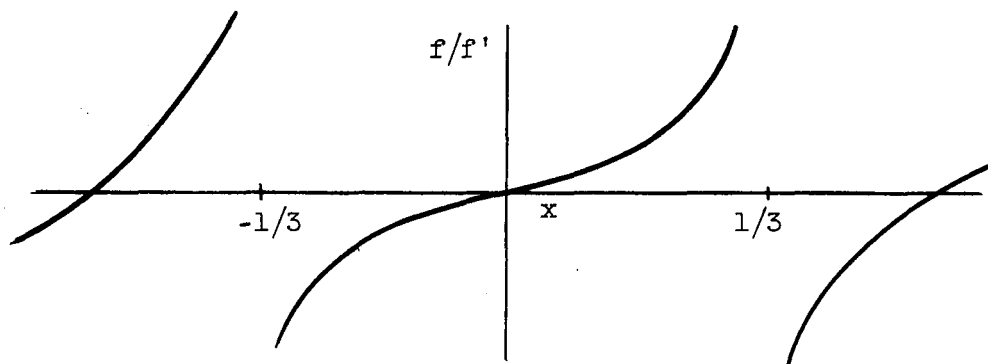
In iterative procedures, since the purpose of an error trace is to obtain an estimate of error, the process must be allowed to continue until both x and E_x have converged. This was demonstrated by using Newton's procedure to obtain the root of $f(x) = x^3 - 3x^2 + 3x - 1$ (triple root at $x = 1$) with a starting value of $x_0 = 0$. The process converged and gave a good estimate of error, but it required a few extra iterations beyond those required for only the principal part to converge. Contrasted to this procedure, by using a modification of Newton's procedure, both convergence and an estimate of error were obtained in one iteration. The modification that was used for this demonstration was to replace $f(x)$ by the test function $f(x)/f'(x)$ in Newton's iterative method, which modifies the algorithms such that

$$x_{k+1} = x_k - \frac{f/f'}{(f')^2 - f/f''}$$

This improved convergence stemmed from the fact that Newton's process acts as a first order process in the vicinity of a multiple root, while the modified Newton's process is a second order process.

Convergence, however, cannot be used as the only basis for obtaining an estimate of error in iterative processes. Conflicts can occur in which the principal part and the error converge to numerically unrelated values. This was demonstrated in using the modified Newton process to obtain the

solution to the equation $3x^3 - x = 0$. The following plot shows the test function (i.e., f/f') to which Newton process is applied to obtain the modified algorithm:



The region of influence of the roots $r_0 = -1/\sqrt{3}$, $r_1 = 0$, and $r_2 = 1/\sqrt{3}$ is $-\infty < r_0 < -1/3$, $-1/3 < r_1 < 1/3$, and $1/3 < r_2 < \infty$, respectively. That is, when a starting value is in a particular region of influence, the process will converge to its associated root. By putting the nominal precision value of the starting value in one region of influence and the nominal precision value plus the error in another region of influence, the algorithm converges to two different roots. In this example, by putting the starting value at $x = 1/3^-$ and $x + E_x = 1/3^+$, the process converged to 0, while the error converged to 0.577.

This same phenomena was further demonstrated by using Newton's algorithm to solve for the roots of $3x^3 - x = 0$. In this case, because of the distributions of the regions of influence, the process could be made to converge to the whole gamut of possibilities with only slight modifications of the low order digits of the starting value. That is,

by using starting values that differ only in the low order digits, this process could be made to converge to $-1/3$, 0, or $1/3$, and to give an estimate of error in each case, and, also, to converge to any combination of two different roots, that is, $-1/3$ and 0, 0 and $1/3$, and $-1/3$ and $1/3$.

CONCLUSION

The error arithmetic presented in this paper is based on higher accuracy calculations and one in which signed errors rather than error bounds are calculated. The method for calculating the errors resulting from a rational calculation has been experimentally verified to produce a realistic estimate of errors. This success has encouraged the consideration of automatically tracing errors resulting from truncation errors. Specifically, a realistic estimate of error can be obtained in the evaluation of a nonalgebraic function if a high accuracy routine is available for evaluating the higher accuracy argument and comparing this to that obtained through the nominal precision arithmetic. A major problem may exist when calculations by two precisions affect decisions in branching on value. In many cases, this will not cause difficulty, but it may in turn be a vital clue to the source of difficulty in an attempted solution of a problem.

Finally, iterative processes were studied with this arithmetic in order to ascertain whether a realistic estimate of error accounting for both rounding and iterative truncation errors could be obtained. The experience has indicated that in most cases this arithmetic could be effectively used as a debugging aid in iterative processes. Cases may arise, however, in which the nominal precision result and the higher precision result converge to two unrelated answers.